IS EVOLUTION ALGORITHMIC?

Marcin Miłkowski

Institute of Philosophy and Sociology

Polish Academy of Sciences

+48226351350

marcin.milkowski@ifispan.waw.pl

Abstract. In *Darwin's Dangerous Idea*, Daniel Dennett claims that evolution is algorithmic. On Dennett's analysis, evolutionary processes are trivially algorithmic because he assumes that all natural processes are algorithmic. I will argue that there are more robust ways to understand algorithmic processes which make the claim that evolution is algorithmic empirical and not conceptual. While laws of nature can be seen as compression algorithms of information about the world, it does not follow logically that they are implemented as algorithms by physical processes. For that to be true, the processes have to be part of computational systems. The basic difference between mere simulation and real computing is having proper causal structure. I will show what kind of requirements this poses for natural evolutionary processes if they are to be computational.

Preliminary draft. Final version available as:
Miłkowski, Marcin. 2009. "Is Evolution Algorithmic?"
Minds and Machines 19 (4): 465–475. doi:10.1007/s11023-009-9170-6.
http://www.springerlink.com/index/10.1007/s11023-009-9170-6.

Daniel Dennett made a claim that evolution is algorithmic (Dennett 1995: 60). Several authors objected that on Dennett's analysis, evolutionary processes could be trivially algorithmic because he assumes that all natural processes are algorithmic (Fodor 1996: 253, Ahouse 1998: 361-363; cf. Dennett 1995: 59). This objection is misleading if all natural processes aren't algorithmic in the sense Dennett wants evolution to be algorithmic. It isn't at all trivial that evolution is algorithmic if all physical processes aren't computational. Pancomputationalism, or universal computationalism is the claim that all physical processes are computational but this, on my strict criteria of computing, will turn out false (for other criticisms of universal computationalism, see Piccinini 2007). The question is how to understand "algorithmic". What would make the claim about the evolution true?

There are processes effectively describable by computations ("algorithmic" in Gregory Chaitin's sense of algorithmic information theory, cf. Chaitin 1975), and processes that realize digital computations. (I am ignoring analog computation here for two reasons. There is no standard analog computation algorithm theory, and the claim I am trying to evaluate is far more controversial when it refers to digital computation.) In what sense are evolutionary processes algorithmic?

All natural processes are algorithmically describable. In this regard, Dennett was right to say they are algorithmic. This is trivial, given the standard algorithmic information theory, though die-hard materialists would probably disagree (see Mahner and Bunge 1997). Yet it's highly controversial that any biological or evolutionary processes are computational. While laws of nature can be seen as compression algorithms of information about the world, it doesn't follow logically that they are implemented as algorithms by physical processes. For that to be true, the processes have to be part of computational systems. The basic difference between mere simulation and real computing is having proper causal structure (Scheutz 2002). Dennett is probably right if he means the weaker claim (evolution can be simulated), and there aren't many reasons to think he's right if he means the stronger, computational claim. That's why Gould (1997) could be right but not because of the reasons that he referred to, as I will show.

Dennett has to defend the specific claim about evolution independently from any claims about all processes, if he didn't mean the first one to be a trivial consequence. However, Dennett's definition of algorithmic processes cannot account for the simulation/computing distinction. Algorithmic processes, according to Dennett, have three features:

1. substrate neutrality
2. underlying mindlessness
3. guaranteed results

The problem is that all functionally describable processes share these features. For example, the process of opening a can be realized using a simple, hand-operated device or engine-driven device, so it's substrate neutral. Opening the can isn't rocket science, either, and it has guaranteed results (*ceteris paribus*). All computational processes are functionally specifiable processes but they have more distinguishing features. A hand-operated can opener isn't a computer, after all. If we fail to see that, we will follow Putnam-Searle fallacy of ascribing any computation to any process: Nothing would disallow ascribing realization of Wordstar program to Searle's notorious wall (Searle 1992: 207-208). Moreover, any disjunction of states of the processes can be thought to realize a computational process, and using arbitrary disjunctions on sufficiently complicated systems, we could ascribe them any possible computation. These results aren't only counterintuitive; they follow from a definitional *fiat* that Searle and Putnam made: They understand computation as a purely syntactic (or formal) object.

Searle argues "syntax is not intrinsic to physics" (Searle 1992: 210). If Searle means that physics isn't linguistics, he's right. Nevertheless, he's wrong to treat algorithms as purely formal syntactic objects. This formalism, if consistent, would make him deny the reality of all mathematical properties ascribed by physicists. It isn't the fact that all mathematical properties ascribed in physics are observer-relative. Properties of computer programs are just the same as the rest. They should be ascribed in the same way science generally ascribes mathematical values to objects.

Computational systems

Contrary to such broad realization concepts, stricter criteria have been recently proposed (see Miłkowski 2006, Scheutz 2001). The list that I'll present is preliminary, and I'll supplement it with a general requirement connected with functional systems as such.

Computational systems are functional systems. There are at least three ways to analyze these systems, according to the notion of function used. First, there is Cummins' notion of function as causal role of a part of the system (Cummins 1975). Second, there are history-based notions, such as defined by Wright (1973) or Millikan (1984). Third, there's a design-based notion of function, as defined by Ulrich Krohs (2004). Cummins' notion is very broad, and makes any causal role a function – for example, the function of the trash can lid is making noise in the middle of the night. According to history-based notions, the function is the cause (a reason) that a thing that has it exists, so prototypes have no functions. Krohs' notion needs a little more explanation.

Krohs suggests that all functional systems have design that specifies system parts in terms of part types. For example, if I want to assemble my IKEA table, I read the manual (the design specs) that specifies the screws, but not as individuals with proper names or located in space-time, but as types. In case of biological systems, the genotype specifies the design. Human-made functional artifacts have parts selected as types by humans; other functional systems are selected by other mechanisms (natural selection seems the most obvious one). This notion has an obvious advantage: the design stance descriptions are literally descriptions based on ascribing design ascriptions. For this very reason, this notion seems appropriate for analyzing Dennett's claims: The design stance would turn out to be based on the notion of design. The task of re-engineering of artifacts and biological systems could be then reformulated as the task of rediscovering their design: their specification in terms of part types and relation of these parts.

Based on these three kinds of notions, three types of functional systems could be defined. The choice of the notion has deeper consequences – probably anything would be a functional system in Cummins' terms but not according to other notions. Prototype systems won't be functional in Millikan's terms, and systems without type-level specifiable parts won't be functional if we accept Krohs' criteria. This means, for example, that dissipative systems which are easily described as wholes in terms of types aren't functional: Their individual parts cannot be picked out using any type-level description—there aren't type-selection mechanisms that would allow for functional ascriptions. Just because dissipative systems are physical systems but not functional systems, they cannot be computational systems, and universal computationalism is false. At the same time, universal computationalism goes hand in hand with Cummins' like functions because parts of dissipative systems could be ascribed causal roles.

The computational description should offer new predictions or explanations. If it isn't the case, the computational description of a given system is redundant, and it's safe to say that the system isn't computational. For example, working of a can opener can be described without stipulating any computation; the can opener doesn't need to process any information about the can to open it (at least that's how today's can openers work). This is just a general rule of stipulating higher-level properties; if lower-level properties are sufficient to predict or explain the behavior and innards of a system, it makes little sense to ascribe higher-lever properties (e.g., it's just as useless to ascribe intentional properties to a lawn). The rule can be spelled out more precisely in terms of Chaitin's algorithmic information theory: the computational description must be simpler than the lower-lever description (a general causal-role level description) and offer epistemic advantages such as new predictions and explanations. The simplicity

boils down to the length of the description (it's equivalent to the compression ratio of the new description compared to the old one). This requirement conflicts with trivial versions of universal computationalism. If universal computationalism could offer new insights for every single physical object, then it would be compatible with the requirement.

The description must be applied consistently for all events in the physical system. We can easily imagine "cheating": devising ascription rules that are far more complex than the system being described, picking out arbitrary disjunctions of states, and so on. This requirement is obvious but notorious "proofs" that any system can perform any computation (Putnam 1987) are so widespread that we should be explicit about the ascription rules. Anyway, that's how natural sciences ascribe mathematical properties, so it shouldn't be controversial.

Ascriptions of sequences of computational states to the system must reflect its causal history. This is just an extension of the consistency requirement into causality. Arbitrary disjunctions of states won't count as causal history so they cannot be described as real computation. This also disallows universal computationalism based only on formal tricks.

The system realizing computations is relatively isolated from the environment. Only functional systems are computational systems, and a system is functional only when it has identifiable boundaries. The boundaries could be blurry but they must delineate the system from the environment. I would define system boundaries in terms of causal relation frequency: causal relations are more frequent inside the system than outside. Even input-output causal relations with a computational system don't make inputs automatically inner values: input relations can obtain with many different objects, which mean that they will be less frequent than real inner relations. If input relations are always connected with the same object or process, this process is a part of the system. This way my delineation criteria help to understand why the notion of extended mind seems intuitive in some cases: it's intuitive only when a remote part of the cognitive system is in fact its subsystem.

It could be argued that some other physical property (other than causal relation relative frequency) should be used to define system boundaries. For example, those who oppose extended mind theories could claim that system boundaries should be spelled out in functional type-level terms of system organization. This kind of system boundary definition is acceptable, as well. What is important is the fact that system boundaries should be definable not only on a computational level of description. Note that arbitrary process state disjunctions nor Searle's wall cannot be clearly delineated on any other level than computational. This poses also another difficulty for universal computationalists because it requires them to show that all physical objects are parts of relatively isolated physical systems.

As I already mentioned, computational systems normally have input states. On the one hand, input data can be internal part of the algorithm the system is implementing. The output data, on the other hand, must be always present. Input and output states should be specifiable, as before, not only on a computational level of description. Note that Searle's wall has no clear input states: there is no wall equivalent of the keyboard nor of the display. Searle hasn't shown any clear way to pick output states nor input states from the set of all states of his wall. There is no computation without output states. Any object can be ascribed a trivially simple output value: Any property could be said to encode it. But this property must be causally related to the input value. So while most objects could be assigned trivial identity transformation (the output property is the input property), non-trivial computations are harder to show.

The input/output requirement is a result of the standard computation definition in terms of recursive functions (as normally Church-Turing thesis is understood). The whole computational process in the system must have a description in terms of recursive function (or any other equivalent model of computation, like Markov strings, Turing machines, register machines etc.). Computational ascription is a real ascription only on the condition that we know what computation we are ascribing. The computation should be spelled out precisely as code or—at least—as pseudo-code.

To sum up, there are several criteria of computational ascriptions:

1. computational description simplicity, predictive and explanatory value
2. description consistency for all processes in the system; causal determination of ascriptions
3. relative system isolation and non-computational boundaries
4. availability of output states connected causally with input states (if any)
5. specification of code-level description

The concept of function realization, which subsumes the realization of computations, depends on how broadly we understand functions. On the design-based notion, pancomputationalism is false. Is Dennett's computational claim false as well?

Evolution as computation

The above top-down analysis of computational systems shows that if there's a real computational level of description of natural evolutionary processes, this cannot be the only level of their description. Could a

computational description of evolutionary processes fulfill the abovementioned criteria?

The computational description will be simpler than the lower-level physical description. Its explanatory value remains, however, at best controversial: It isn't at all clear what it would explain. Origin of biological information as selected from the chaos? Or the way natural selection works? The predictive value isn't clear neither. Whereas the general algorithm of evolution could predict the way natural selection works in every case, it would probably be highly dependent on the complete knowledge of environmental constraints and details of evolution units being selected. It isn't clear that these predictions wouldn't be available in the modern neo-Darwinian Synthesis. For the sake of argument, let us suppose that we would gain an insight into how evolution, or Mother Nature in Dennett's terms, processes information about replicators and interactors (Brandon 1998).

We would apply the description consistently, based on causal relations. Therefore, we assume that consistency requirement would be fulfilled.

Evolutionary processes are probably relatively easy to single out from other processes (say, geological) but it isn't obvious whether the most relevant elements of these processes have any function in the evolutionary computational systems. Are evolutionary processes like dissipative processes? According to the more robust, design-based notion of function, physical processes can implement algorithms but not all kinds of physical processes are computational: Those that form non-linear and non-aggregative systems that strongly depend on token-only properties like space-time localization cannot have functional elements. If it could be shown that the way evolutionary processes run depends only on their localization (or any purely token-level property), Dennett's claim would be false. At the first glance, this is what Gould wants us to believe:

> Crank your algorithm of natural selection to your heart's content, and you cannot grind out the contingent patterns built during the earth's geological history. You will get predictable pieces here and there (convergent evolution of wings in flying creatures), but you will also encounter too much randomness from a plethora of sources, too many additional principles from within biological theory, and too many unpredictable impacts from environmental histories beyond biology (including those occasional meteors)—all showing that the theory of natural selection must work in concert with several other principles of change to explain the observed pattern of evolution (Gould 1997).

Gould thinks that contingency – responsible for all variability of the population – plays such an important role in natural selection that its algorithm cannot be realistic without considering this contingency. However, contingencies, or initial state of the environment fed into the computational evolutionary process can be treated in two ways: first, they can be described using lossy compression, and second, simply be input into the more complex computational system. Both ways are compatible

with a notion of algorithm realization. What Gould hasn't shown is that these contingencies would make the natural selection algorithm computationally intractable because of the combinatorial explosion.
To answer the question whether it would be computationally tractable, we need the code. What should this code compute? A general natural selection problem or a specific selection problem? According to Gould, we could produce an algorithm for convergent evolution, so this could be a third possibility.
Let's start with the first possibility: a general natural selection algorithm. The fitness of units being selected naturally shows that the solution of the problem of adapting to environment was effectively solved. I would propose that the evolutionary algorithm solves the problem of adaptation, and this fitness or the adapted population could be thought of as output value of the computation. Maybe the interaction with environment could produce the input of this algorithm.
Some hints about what evolutionary computational systems are and what kind of computations they realize can be found in computer science. Research on artificial life or evolutionary algorithms seems to suggest that though there are emergent properties and strong context-dependence of properties, at the same time objects are computational (Crutchfield and Mitchell 1995). It's an empirical question whether natural evolutionary processes are like dissipative systems or rather like Artificial Life.
Evolutionary algorithms are heuristic search algorithms modeled after natural processes (Michalewicz 1996). They involve generating and mutating a population of artificial organisms, and testing them according to a fitness function. The fitness is assessed based on how well the given organism finds a solution to a problem. There are various types of evolutionary algorithms, and not all properties and types of evolutionary algorithms are now known. Most likely, existing evolutionary algorithms are only a small subclass of all possible evolutionary algorithms. Compared to natural processes, artificial ones are less complex but can serve as a starting point for evaluating Dennett's claim.
The problem with the code inspired by the research on evolutionary algorithms is that it cannot be adopted directly. The overall structure of evolutionary algorithms is as follows:

```
procedure evolution
begin
   t=0
   determine_starting_P(t)
   (* P(t) - Population P at time t *)
   final_condition=(evaluate P(t) >threshold)
   while not (final_condition) do
   begin
      t=t+1
      select P(t) (* from P(t-1) *)
      modify P(t)
      final_condition=(evaluate P(t) >threshold)
```

```
   end
end
```

The problem is that this algorithm is based on the *evaluate()* function. This function is however encoded by the programmer, not discovered by the algorithm itself. No general algorithm of fitness assessment seems viable, though here various flavors of adaptationism could have their say. In nature, the encoding of the fitness function is unknown. The fitness landscape is not represented numerically in reality. Therefore, the straightforward application of such algorithms results in a very unsatisfactory code:

```
procedure evolution
begin
t=0
fix starting P(t)
while not (the_end_of_the_world) do
        begin
        t=t+1
        select_naturally_in_environment P(t)
        modify P(t)
        end
end
```

This code cannot possibly fulfill the requirement of explanatory value. It isn't giving any new predictions, and seems only a trivial and redundant reformulation of known causal mechanisms of natural selection.
Replacing the explicit fitness function representation with ways of discovering the environmental constraints might be one of the ways out of this problem. Natural evolutionary computation cannot represent fitness functions that merely make it easier to humans to simulate the causal relations between populations and their environment.
However, it might be argued that general code structure is, in principle, always sketchy and trivial. What we need to find is a specific code for specific evolutionary processes. It might be inspired by current research on evolutionary algorithms or not. In other ways, we should use the bottom-up method to try to find the code.
So I turn to the second possibility, namely to the code computing a specific natural selection process. The arms race between bacteria and antibiotics has been simulated "in silico" by various researches. It's even possible to use "in silico" models to discover new drugs (Gray and Keck 1999). These models start with bacteria genome sequences and proteins expressed by genomes, and knowledge about genes that are crucial for the survival of the bacteria. Such massive data can be then used to predict if certain bacteria or their mutation can survive at all. The general structure of the first algorithm could be used, with appropriate substitutions. The *evaluate* function would test if the bacteria survives when certain proteins are destroyed; it wouldn't require a separate representation.

Does it allow us to say that computational simulation of bacteria vs. antibiotics is really algorithm realization? Not at all. Current "in silico" methods often use data gathered from parallel *in vivo* experiments because scientists still don't know what's being ignored in the simulation. The simulation doesn't include all the causal-functional details, and some of them probably should be disregarded for the sake of simplicity in many cases. Nevertheless, the detailed "in silico" experiment could, in principle, fulfill all criteria of computation realization.

What about convergent evolution? Gould clearly sees that there is a regular pattern in the evolution of the wing in many species. The evolution of the wing could be regarded as an engineering problem, requiring optimization methods. Evolutionary algorithms are used for airplane wing optimization (Keane and Petruzzelli 2000), and a recombination of the existing wing optimization with special organic wing requirements would give us an algorithm for selecting the wing in many different species. The fitness function would be based on aerodynamic features and general engineering principles.

Specific algorithms aren't prone to the problem of how to evaluate the fitness generally. Yet all three sketched algorithms share another disadvantage. They are simulation algorithms rather than algorithms of natural information processing. It cannot be proved that there isn't any other algorithm for natural selection in play, as the problem of existence of any algorithm *for* something is itself not decidable in general: the only way to prove that there is an algorithm for something is to show it.

Biologists, even computational biologists, generally don't seek for computational structure in natural selection. They either build artificial computational systems with biological parts or simulate biological processes. This could mean that, after all, Dennett was right that there are algorithmic processes *in vivo*. But only in Chaitin's sense of the term. Dennett's definition of algorithmic process is redundant then, and reduces easily to the Chaitin's technical term. That's why it doesn't account for simulation/computation distinction. Simulation can sometimes produce a genuine article, for example in a simulated theorem prover. However, in case of evolutionary algorithms it's only a description of evolutionary processes that they produce, and not adapted populations. Moreover, a description shouldn't be confused with what it describes.

Algorithm, natural law, real pattern?

So maybe Dennett's claim isn't about computational powers of evolution but rather about real, multiple-realizable patterns of evolution. These patterns are algorithmic in Chaitin's sense: they aren't stochastic, and there is a way to see regularity in them. After all, the biological data is not all but noise.

A vague usage of “algorithm” is often found in biological papers. Manfred Eigen writes “Our task is to find an algorithm, a natural law that leads to the origin of information” (Eigen 1992, 12). As Mayr notes, biologists often use “models”, “algorithms”, “theories”, “conjectures” interchangeably (Mayr 1997). The problem with this usage, which is roughly compatible with Chaitin’s notion of the algorithmic, is that nature doesn’t realize algorithms for describing the natural processes, and Chaitin’s algorithms are algorithms for describing sequences of information. Therefore, while a process could be algorithmic in Chaitin’s sense, there could be no algorithm that it implements. The implementation could be found somewhere else, for example in a human observer. In other words, evolutionary processes are algorithmic in this sense but aren’t necessarily doing any computations whatsoever.

This notion of the algorithmic is completely compatible with multiple realizability and substrate neutrality. In short, all functionally specifiable processes are algorithmic in Chaitin’s sense, and algorithmic descriptions could be re-used (as non-token level descriptions) to refer to potentially many objects, also made of some other stuff. This leads to a conclusion that multiple realizability of natural selection could be still maintained but natural selection would be no algorithm, only a process in functional systems. But would anyone try to argue with it?

The weaker reading of Dennett’s claim would still face resistance. In principle, functional systems could include cultural processes – as suggested by memetics – or various units of natural selection. There are materialists who claim that it is stuff that matters, and they would object that natural selection isn’t multiple realizable nor functional (Mahner and Bunge 2001). However, this notion of algorithmic processes doesn’t involve any implementation of formal properties or “syntax” by natural selection and this is the premise on which their objection depends, just like in the case of Searle.

Yes, natural selection processes are lawful and not stochastic. They are real patterns. Yes, this is trivial. And it’s much more interesting to see their functional structure in specific cases rather than to say they’re generally algorithmic.

References

Ahouse, J. C., 1998, “The Tragedy of a priori Selectionism: Dennett and Gould on Adaptationism,” *Biology & Philosophy*, 13, 359-391.

Brandon, R., 1998, “The Levels of Selection: A Hierarchy of Interactors,” in: Hull D. and M. Ruse, *The Philosophy of Biology*, Oxford University Press, p. 176-197.

Chaitin, G. J., 1975, “Randomness and Mathematical Proof,” *Scientific American,* 232, No. 5, 47-52.

Crutchfield, J. P. and M. Mitchell, "The Evolution of Emergent Computation", *Proceedings of the National Academy of Sciences, USA* 92:23 (1995), 10742-10746.

Cummins, R., 1975, “Functional Analysis,” *The Journal of Philosophy*, 72, No. 20, 741-765.

Dennett, D., 1995, “Darwin’s Dangerous Idea: Evolution and the Meanings of Life,” Simon & Schuster, Nowy Jork.
Eigen, M., 1992. *Steps toward Life: A Perspective on Evolution*. Oxford: Oxford University Press.
Fodor, J., 1996, “Deconstructing Dennett’s Darwin,” *Mind & Language*, 11, No. 3, 246-262.
Gould, S. J., 1997, "Evolution: The Pleasures of Pluralism," *The New York Review of Books*, June 26, 1997, 47-52.
Grey, C.P. and W. Keck, “Bacterial targets and antibiotics: genome-based drug discovery,”, *Cellular and Molecular Life Sciences*, 56 (1999), 779-787.
Keane, A. J. and N. Petruzzelli, "Aircraft wing design using GA-based multi-level strategies", pp. A00-40171 AIAA-2000-4937 in *Proceedings of the 8th AIAA/USAF/NASSA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, A.I.A.A., Long Beach (2000).
Krohs, U., 2004, *Eine Theorie biologischer Theorien*, Springer Verlag, Berlin.
Mahner, M. and M. Bunge, 2001, “Function and Functionalism: A Synthetic Perspective,” *Philosophy of Science*, Vol. 68, No. 1, 75-94.
Mayr, E., 1997, “This is Biology. The Science of the Living World,” Belknap Press.
Michalewicz, Z., 1996, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag.
Miłkowski, M., 2006, “Is computationalism trivial?,” In Gordana Dodig Crnkovic and Susan Stuart (eds.), *Computing, Philosophy, and Cognitive Science*, Cambridge Scholars Press (forthcoming).
Millikan, R.G., 1984, *Language, Thought, and Other Biological Categories. New Foundations for Realism*, Cambridge. MA: MIT Press.
Piccinini, G., 2007 “Computational Modelling vs. Computational Explanation”, *The Australasian Journal of Philosophy* (forthcoming).
Putnam, H., 1987, *Representation and Reality*. Cambridge, MA: MIT Press.
Scheutz, M., 2001, “Computational versus Causal Complexity,” *Minds And Machines*, *11*, 543-566.
Scheutz, M., 2002, “Philosophical Issues about Computation”. In *Encyclopedia of Cognitive Science*, London, UK. MacMillan Publishers.
Searle, J., 1992, “Rediscovery of Mind,” MIT Press, Cambridge (Mass).
Wright, L., 1973, “Functions,” *The Philosophical Review*, 82, No. 2, 139-168.